Creating PLT-Style Web Pages

Version 6.1

August 1, 2014

`#lang plt-web` package: `plt-web-lib`

The `plt-web` language builds on `scribble/html` for generating pages in the same style as `http://racket-lang.org/`.

Unlike `scribble/html`, the values of expressions in a `plt-web` module are not treated as HTML. Instead, top-level expressions in `plt-web` are as in `racket/base`.

Meanwhile, a `main` submodule is added that runs `render-all` (after parsing command-line arguments) to render all registered resources, including HTML pages. Pages are meant to be registers through a `page` form that is defined by a `define-context` or `define+provide-context` declaration that configures a particular site (i.e., collection of pages).

1 Configuring a Site

A site is identifiers by a relative directory path, which determines where the site's content is generated. For a non-local build, the relative directory is mapped to a destination URL via `url-roots`.

```
(site dir
      [#:url url
       #:always-abs-url? always-abs-url?
       #:generate? generate?
       #:share-from share-from
       #:page-style? page-style?
       #:page-headers page-headers
       #:meta? meta?
       #:robots robots
       #:htaccess htaccess
       #:navigation navigation])          → site?
  dir : path-string?
  url : (or/c string? #f) = #f
  always-abs-url? : any/c = #t
  generate? : any/c = #t
  share-from : (or/c site? #f) = #f
  page-style? : any/c = #t
  page-headers : outputable/c = null
  meta? : any/c = page-style?
  robots : (or/c #f #t outputable/c) = #t
  htaccess : (or/c #f #t outputable/c) = #t
  navigation : (listof outputable/c) = null
```

Creates a value that represents a site.

If *url* is not `#f`, then it will be registered to `url-roots` for a build in web mode (as opposed to local mode) and recorded as the target for *dir* in a `"sites.rktd"` file when building in deployment mode.

If *always-abs-url?* is true (the default), then *url* is registered with a `'abs` flag, so that (in deployment mode) references within a site are relative to the site root, as opposed to relative to the referencing resource.

If *generate?* is `#f`, then resources for the site (such as icons or CSS files) are not generated.

If *share-from* is a site, then resources generated for the site (such as icons or CSS files) are used when as possible for the new site.

If *page-style?* is true, HTML pages generated for the site include content to set the style of the overall page. Otherwise, only sufficient resources and content are included to specify

the style of the PLT web-page header (i.e., a bar with the Racket logo).

The *page-headers* argument provides content for the header of any HTML page generated at the site via page or page*.

If *meta?* is true, then ".htaccess", "robots.txt", and similar files are generated for the site. The *robots* and *htaccess* arguments determine robot and access information included by the default resource-mapping function. A #t value enables normal access, a #f value disables access, and any other value is used as the corresponding specification.

The *navigation* argument determines content (usually links) to be included in the PLT header. Currently, up to four such links are supported.

```
(site? v) → boolean?
  v : any/c
```

Returns #t if *v* represents a site, #f otherwise.

```
(site-dir s) → path-string?
  s : site?
```

Extracts the destination directory of *s*.

```
(site-css-path s) → outputable/c
  s : site?
```

Extracts a reference to a CSS resource for HTML pages at site *s*.

```
(site-favicon-path s) → outputable/c
  s : site?
```

Extracts a reference to a "favicon.ico" file for the site *s*. The result is #f if meta-file resources are not generated for the site.

```
(site-navbar s) → outputable/c
  s : site?
```

Generates HTML for the banner on HTML pages at the site *s*.

```
(site-navbar-dynamic-js s) → outputable/c
  s : site?
```

Generates a JavaScript definition of a AddNavbarToBody function, which adds a banner dynamically to the current page for a page at site *s*.

2 Creating Site Content

```
(page keyword-arg ... form ...)

keyword-arg = keyword expr
```

Equivalent to `(page* keyword-arg ... (lambda () (begin/text form ...)))`.

```
(page*  #:site s
        [#:html-only html-only?
         #:id id
         #:file file
         #:title title
         #:link-title link-title
         #:window-title window-title
         #:width width
         #:description description
         #:extra-headers extra-headers
         #:extra-body-attrs body-attrs
         #:referrer referrer
         #:part-of part-of]
         content)                          → outputable/c
  s : site?
  html-only? : any/c = #f
  id : path-string? = #f
  file : (or/c path-string? #f) = #f
  title : string? = (... id)
  link-title : outputable/c = title
  window-title : string? = (string-append "Racket: " label)
  width : (or/c #f 'full outputable/c) = #f
  description : string? = #f
  extra-headers : outputable/c = #f
  body-attrs : outputable/c = #f
  referrer : (string? outputable/c ... . -> . outputable/c)
           = (λ (url . content)
               (a href: url (if (null? content) linktitle content)))
  part-of : (or/c #f symbol?) = #f
  content : outputable/c
```

Registers an HTML page as a resource (via `resource`) and returns a value that can be used to refer to the page within another resource, assuming that *html-only?* is `#f`. If *html-only?* is true, then the result represents HTML for the page, instead of a way to reference the page, and no resource is registered.

The page is generated as part of the site *s*, and either an *id* or *file* must be provided to identify the page within the site. Furthermore, either *id* or *title* must be provided to

determine the page's title.

The *link-title* and *window-title* arguments control separate the title of the page as used by references and for the page as viewed.

The *width* argument determines the page wide: `#f` is the default, `'full` is full width, and any other value is used as a CSS width.

The *description* argument provides a meta tag for the page.

The *part-of* argument determines where the page resides in a larger site when the layout uses a global navigation bar (but the current format does not use a navigation bar in that sense).

```
(plain keyword-arg ... form ...)

keyword-arg = keyword expr
```

Equivalent to `(plain* keyword-arg ... (lambda () (begin/text form ...)))`.

```
(plain*  #:site s
         [#:id id
          #:suffix suffix
          #:file file
          #:referrer referrer
          #:newline newline?]
          content)            → outputable/c
  s : site?
  id : path-string? = #f
  suffix : (or/c #f string?) = #f
  file : (or/c path-string? #f) = #f
  referrer : (string? outputable/c ... . -> . outputable/c)
           = (λ (url . content)
               (a href: url (if (null? content) linktitle content)))
  newline? : any/c = #t
  content : outputable/c
```

Like page*, but for a resource that is a plain file.

```
(copyfile #:site s src [dest]) → outputable/c
  s : site?
  src : path-string?
  dest : string? = (basename src)
(symlink #:site s src [dest]) → outputable/c
  s : site?
  src : path-string?
  dest : string? = (basename src)
```

Registers a resource that is either a copy of a file or a symbolic link, returning a value that can be used to reference the resource.

```
(make-indexes  s
               dir
              [#:depth depth
               #:use-dir? use-dir?]) → void?
  s : site?
  dir : (or/c 'same relative-path?)
  depth : (or/c #f exact-nonnegative-integer?) = #f
  use-dir? : ((or/c 'same relative-path?) . -> . any/c)
           = (lambda (dir) #t)
```

Uses index-site and index-page to register an "index.html" file for every directory within *dir* (relative to the current directory) that does not have an "index.html" file already. If *depth* is not #f, then subdirectories are explored at most *depth* layers deep. The *use-dir?* predicate is called for each directory to determine whether the directory's subdirectories are traversed.

The generated index files are registered for the site *s* at destinations that correspond to treating the current directory as the site root.

```
(index-site? v) → boolean?
  v : any/c
(index-site site) → index-site?
  site : site?
(index-page  isite
             path
             content
            [#:html-only? html-only?]) → outputable/c
  isite : index-site?
  path : (or/c 'same relative-path?)
  content : (listof (cons/c path-string? (or/c exact-integer? 'dir)))
  html-only? : any/c = #f
```

The index-page function registers an individual "index.html" file (or returns its content if *html-only?* is true) for the given index site, where an index site is created once for a given site (to register support resources, such as icons). The "index.html" file is generated for the subdirectory indicated by *path*. The index file lists the content specified by *content*, where an integer corresponds to a file size and 'dir indicates a directory.

```
(call-with-registered-roots thunk) → any
  thunk : (-> any)
```

Calls *thunk* with url-roots set to a mapping for registered sites.

3 Generating Site Content

To generate web pages from a `plt-web` module, run the module, typically with a `-o` flag to specify the output directory. For example, if `"pages.rkt"` is the module, then

```
racket pages.rkt -o /tmp/pages
```

builds the output to a `"/tmp/pages"` directory.

The command-line flags are recognized by the `main` submodule that is introduced by `plt-web`:

- `-w` or `--web` — Build output in *deployment mode*, where references within a top-level site use relative paths or site-relative paths (that start with `/`), while references across top-level sites use absolute URLs. This mode is the default.
- `-l` or `--local` — Build output in local mode using `"file://"` URLs between top-level sites and relative paths within a site.
- `-r` or `--relative` — Build output in local-relative mode, where all references use relative paths, exploiting the fact that sites are rendered in adjacent directories within the output directory. (You may need to deal with an occasional manual selection of `"index.html"` when viewing relative output.)
- `-o` ⟨*dir*⟩ or `--output` ⟨*dir*⟩ — Writes output to subdirectories of ⟨*dir*⟩, which defaults to the current directory. All existing files and directories within ⟨*dir*⟩ will be deleted. As a safety check, the destination directory must overlap with any installed package directory.
- `-f` or `--force` — Overwrite files in the destination directory.
- `+e` ⟨*path*⟩ or `++extra` ⟨*path*⟩ — Require the module ⟨*path*⟩ before running `build-all`. This flag can be used multiple times.

4 Utilities

```
(basename p) → string?
  p : path-string?
```

Extracts a file name from a path.

```
(web-path str ...) → string?
  str : string?
```

Joins *str*s with a "/" separator to form a relative URL path.